ECSE 429 - Project Part 1 Report

by David Vo and David Zhou

1. OVERVIEW

This project focused on the exploratory and unit testing of a REST-based Todo Manager API. The system provides functionality for managing todos and supports extended relationship endpoints linking todos to categories and projects. The goal of this work was to discover documented and undocumented capabilities, evaluate system robustness, identify potential stability risks, and design a supporting unit test suite.

Two exploratory testing sessions were conducted. The first session focused on understanding and validating the core /todos CRUD functionality. The second session extended this exploration to the newer /todos relationship endpoints and robustness testing. Based on these sessions, scripts and standalone JUnit tests were developed to demonstrate discovered capabilities and to provide automated verification.

This report summarizes the project deliverables, documents the findings from exploratory testing, describes the structure of the unit test suite and repository, and presents the results of executing the automated tests.

2. DELIVERABLE SUMMARIES

The following deliverables were produced:

- Two time-boxed exploratory testing sessions with detailed session notes.
- A scripted execution file using curl to demonstrate discovered behaviors and undocumented features.
- A standalone JUnit-based unit test suite targeting both documented and undocumented behaviors.
- Session artifacts including command logs and screenshots.
- This written report summarizing findings, risks, and test structures.

Each deliverable was designed to support the others: exploratory testing identified risks and behaviors, scripts demonstrated them, and unit tests automated their verification.

3. EXPLORATORY TESTING FINDINGS

3.1 Session 1 - Core /todos Capabilities

The first session focused on understanding baseline functionality. Core CRUD operations were exercised, including:

- GET /todos
- POST /todos
- GET /todos/:id
- POST /todos/:id
- PUT /todos/:id
- DELETE /todos/:id

The API supported creation, retrieval, update, and deletion of todos as documented. Input validation was partially enforced. For example:

- Missing required fields (title) returned 400 errors.
- Empty titles failed validation.
- Extra fields were rejected.
- Deleting nonexistent resources returned 404 responses.

However, differences between POST and PUT behavior were observed. POST updates allowed partial updates, while PUT updates enforced full object validation. These differences were not clearly documented and represent potential usability and consistency concerns.

The API also supported both JSON and XML formats. While correct use worked as documented, malformed input produced raw Java exception messages, exposing internal details.

3.2 Session 2 - Extended /todos Endpoints and Robustness

The second session focused on newly added /todos endpoints and undocumented behavior discovery.

New /todos capabilities identified

- /todos/:id/categories
- /todos/:id/task-of

These endpoints returned related category and project data. On startup, todos were already linked to categories and projects, demonstrating preconfigured relationships.

Relationship deletion endpoints functioned as expected:

- /todos/:id/categories/:id
- /todos/:id/task-of/:id

However, attempts to create relationships using POST with an ID were rejected with validation errors, contradicting the API documentation. This revealed a discrepancy between documented and actual behavior.

Undocumented behaviors discovered

Several important undocumented behaviors were identified:

1. Content-Type header not enforced
   - JSON requests were accepted even when Content-Type was set to application/xml.
   - JSON requests were accepted with no Content-Type header at all.
2. Internal exceptions exposed
   - Sending XML with a JSON content-type returned raw Java parsing exceptions.
3. Unsupported HTTP methods mishandled
   - PATCH and TRACE returned HTML error pages rather than JSON API responses.
4. Silent relationship deletion
   - Relationship deletion returned no confirmation body.
5. Documentation contradictions
   - Relationship POST endpoints existed but did not allow documented usage.

These behaviors indicate weaknesses in protocol validation, error handling, and documentation accuracy.

4. SUMMARY OF EXPLORATORY TESTING

The Todo Manager API supports a wide range of functionality and exposes rich relationship endpoints. Core todo management is stable under normal conditions, but robustness testing uncovered several critical concerns.

The API does not strictly validate request headers, exposes internal exceptions, and inconsistently formats error responses. Relationship behavior contradicts documentation and does not support client-side creation as described. Unsupported HTTP methods return HTML pages, breaking API consistency.

Overall, the system functions under ideal use but demonstrates fragility under malformed or unexpected conditions.

5. UNIT TEST SUITE STRUCTURE

5.1 Design Approach

The unit test suite was implemented as standalone JUnit 5 tests, independent of Maven or Gradle. Tests communicate directly with the running REST service using Java's HttpClient.

The test suite was designed to:

- Validate core /todos behaviors.
- Verify discovered undocumented behaviors.
- Ensure the service is required for test success.
- Demonstrate automated checking of robustness conditions.

Tests are black-box service tests and do not rely on internal source code access.

5.2 Test Organization

The suite is organized into logical groups:

Core API tests

- Create todo (valid input)
- Retrieve all todos
- Retrieve todo by ID
- Delete todo
- Update behavior (POST vs PUT)
- Error handling (missing fields, malformed JSON)

Undocumented and robustness tests

- JSON accepted with XML content-type
- JSON accepted without content-type
- XML with JSON content-type fails
- Unsupported HTTP methods return errors
- Relationship POST restriction
- Relationship deletion behavior

Each test is atomic and independent. The tests rely on HTTP status codes and response body validation.

5.3 Execution Method

Tests are compiled and run using the JUnit platform console:

```
javac -cp junit.jar *.java
java -jar junit.jar --class-path . --scan-class-path
```

The service must be running for tests to pass. When the service is shut down, tests consistently fail, demonstrating correct dependency on the API.

6. SOURCE CODE REPOSITORY DESCRIPTION

The repository contains:

- Exploratory session scripts (.sh)
- Standalone JUnit test files (.java)
- Compiled test classes
- Session notes
- Report documentation

The structure is intentionally lightweight to allow easy execution without external build tools. Scripts reproduce exploratory results. Unit tests provide repeatable verification.

7. UNIT TEST EXECUTION FINDINGS

When the API server is running:

- All core functionality tests pass.
- Relationship retrieval tests pass.
- Robustness tests confirm undocumented behaviors.
- Negative tests consistently detect malformed requests.

When the API server is shut down:

- All service tests fail due to connection errors.
- This confirms tests are exercising the real system and not false positives.

The test results support the exploratory findings:

- Content-type is not enforced.
- Relationship POST restrictions exist.
- Unsupported methods are mishandled.
- Java exceptions leak through the API.

These automated results provide repeatable evidence of system weaknesses and confirm the stability of core operations.

8. CONCLUSION

This project demonstrated the value of exploratory testing combined with automated verification. Exploratory sessions revealed undocumented behaviors, protocol weaknesses, and documentation mismatches that would not be detected through scripted functional testing alone.

The unit test suite formalized these findings into repeatable checks and validated both expected and unexpected system behavior. The results indicate that while the Todo Manager API supports its core features reliably, it lacks robust protocol enforcement and consistent error handling.

Future work should prioritize improving documentation accuracy, sanitizing error responses, enforcing headers, and strengthening relationship endpoint behavior.